IT LAB

Инструкция по пуску

Интерактивная аналитическая панель

СОДЕРЖАНИЕ

1 ВВЕДЕНИЕ 2

1.1 Список изменений 2

2 СПИСОК СОКРАЩЕНИЙ 3

3 НЕОБХОДИМЫЕ УСЛОВИЯ 4

3.1 Требования к квалификации специалиста 4
3.2 Комплект поставки 4
3.3 Требуемое оборудование и ПО 4

4 ПУСК ПРИЛОЖЕНИЯ 5

4.1 Предварительная подготовка сервера 5
4.1.1 Установка ПО 5
4.1.2 Установка БД Postgres через Docker Compose 5
4.1.3 Установка Mosquitto через Docker Compose 6
4.2 Установка демонстрационного приложения 7
4.3 Передача данных в демонстрационное приложение 8

5 НАСТРОЙКИ СИСТЕМЫ 12

5.1 Шаблонные объекты 12
5.2 Спецификация полей entity 13
5.3 Пользовательский интерфейс 15
5.3.1 Верхний заголовочный блок 16
5.3.2 Левая панель 18
5.3.3 Страницы 19
5.3.4 Карточки 20
5.3.5 Настройки системы 26

6 ИСПОЛЬЗОВАНИЕ КОМПОНЕНТОВ СИСТЕМЫ ДЛЯ ВСТРАИВАНИЯ В СВОИ ПРИЛОЖЕНИЯ 28

6.1 Интеграция как сервис 28
6.2 Интеграция в Spring Boot приложение 28
6.2.1 Базовая интеграция в Spring Boot 29
6.2.2 Интеграция в приложение React 30

7 ПРИЛОЖЕНИЕ 1. КОНФИГУРАЦИЯ ДЕМОНСТРАЦИОННОГО ПРИЛОЖЕНИЯ 32

1 ВВЕДЕНИЕ

В данной инструкции описывается процесс установки, настройки и запуска приложения Интерактивной аналитической панели, в дальнейшем, для краткости, называемого «система».

Интерактивная аналитическая панель представляет собой компонент, который с помощью динамически обновляемого web-интерфейса в компактной и удобной форме отображает параметры, получаемые от внешнего источника. В качестве источника данных для отображения (по сути, входного протокола системы) в текущей версии выступает MQTT-сервер, на котором в определенных топиках публикуются сообщения с обновленными значениями параметров. Система позволяет производить настраиваемые преобразования параметров перед отображением, например, вычленить из опубликованного в MQTT сообщения в формате JSON какое-то определенное поле, произвести над значением какие-либо математические или логические преобразования и т.п.

Система сохраняет в базе данных историю изменения полученных параметров и дает возможность отобразить ее на интерфейсе.

Для демонстрации возможностей системы на ее базе собрано демонстрационное приложение для отображения счетчиков умного дома. На его примере в данной инструкции рассмотрены варианты использования системы. Приложение можно настроить и подключить в качестве информационной панели в составе умного дома, базирующегося на HomeAssistant (https://www.home-assistant.io) с включенным MQTT-расширением, либо для визуального контроля состояния сети устройств, подключенных к Zigbee2mqtt (https://www.zigbee2mqtt.io).

1.1 Список изменений

Версия документа	Дата	Изменение
1.0	01.09.2023	Исходная версия документа

2 СПИСОК СОКРАЩЕНИЙ

Сокращение	Расшифровка
ОС	Операционная система
ПО	Программное обеспечение
БД	База данных

3 НЕОБХОДИМЫЕ УСЛОВИЯ

3.1 Требования к квалификации специалиста

Все операции, описанные в данной инструкции, должны выполняться специалистом, обладающим следующими квалификациями:

- установка и настройка ОС Linux;
- представления о протоколе MQTT и структуре публикации сообщений на MQTT-сервере;
- представления о формате JSON, его устройстве и назначении;
- владение языком разметки Jinja2 (https://jinja.palletsprojects.com/en/3.1.x/templates/).

3.2 Комплект поставки

В комплект поставки входят следующие файлы:

- `uitabs-demo-app-x.x.x.jar` – дистрибутив приложения.
- `uitabs-core-x.x.x.jar` – базовая библиотека функционала системы.
- `uitabs-backend-x.x.x.jar` – библиотека для встраивания системы в приложения Spring Boot.
- `uitabs-ui-x.x.x.tgz` – библиотека для встраивания системы в React-приложения.
- `UITABS-GL-RU-00.00.00.dSTP.01.00.pdf` – текущая инструкция по пуску.

3.3 Требуемое оборудование и ПО

Для установки системы требуется сервер или виртуальная машина с двумя доступными процессорными ядрами, 4Gb RAM и не менее 40 Gb HDD/SSD хранилища.

В рамках данной инструкции на сервер будет установлена ОС Ubuntu 22.04 LTS. Это не является обязательным требованием, возможно использование любого дистрибутива Linux, да и вообще любой ОС, на которой возможен запуск Java 17 (или выше) и БД PostrgeSQL.

Предполагается, что источник отображаемых данных в виде сервера MQTT уже установлен и запущен. Установка и настройка сервера MQTT не входит в рамки данной инструкции. В качестве сервера MQTT может использоваться широко распространенный Eclipse Mosquitto (https://mosquitto.org), для которого существует большое количество документации и инструкций по установке и настройке.

4 ПУСК ПРИЛОЖЕНИЯ

4.1 Предварительная подготовка сервера

4.1.1 Установка ПО

Необходимо установить на сервер следующее ПО:

- **Ubuntu 22.04** – Дистрибутив и инструкцию по установке можно найти по ссылке: https://help.ubuntu.com/community/Installation
- **Docker Engine** - Установите на сервере сайта. Дистрибутив и инструкцию по установке можно найти здесь: https://docs.docker.com/engine/install/ubuntu
- **Java 17** - Для установки выполните команду

```
$  sudo apt-get install openjdk-17-jdk
```

Настройте операционную систему на сервере: обеспечьте себе удаленный доступ по ssh, добавьте привычный текстовый редактор, мультиплексор экранов. В общем, сделайте так, чтобы вам было удобно.

4.1.2 Установка БД Postgres через Docker Compose

Безусловно, можно установить PostgeSQL непосредственно на сервер или использовать уже установленный там экземпляр. Но для быстрого запуска мы будем использовать docker-контейнер.

1. В какой-нибудь подходящей директории (например, в `/opt`) создайте файл `docker-compose.yml`. Очевидно, вам для этого понадобится root-доступ. Вообще, все операции с docker-ом придется производить с помощью `sudo`.
2. Вставьте туда содержимое:

```
version: '3.3'

services:
  postgres-uitabs:
    # Или укажите образ той версии Postgres, которая вам нравится.
    image: postgres
    environment:
      - TZ=Europe/Moscow
      - POSTGRES_DB=${POSTGRES_DB}
      - POSTGRES_USER=${POSTGRES_USER}
      - POSTGRES_PASSWORD=${POSTGRES_PASSWORD}
      - PGDATA=/var/lib/postgresql/data/pgdata
    volumes:
      - uitabs-data:/var/lib/postgresql/data/
    ports:
      - "5432:5432"
    restart: unless-stopped

volumes:
  uitabs-data:
```

3. Создайте файл `.env` рядом с `docker-compose.yml` для того, чтобы хранить доступы к БД не в `docker-compose.yml`, а отдельно. Укажите логин и пароль, которые вам нравятся.

```
POSTGRES_DB=uitabs
POSTGRES_USER=uitabs
POSTGRES_PASSWORD=mysecurepassword
```

4. Запустите докер

```
$  sudo docker compose up -d
```

После запуска docker скачает текущую версию контейнера и запустит его. У вас на локальном порту 5432 есть готовая БД, которая вас слушает.

4.1.3 Установка Mosquitto через Docker Compose

Если у вас в области видимости уже есть развернутый MQTT-сервер, то вы можете подключиться к нему. Нужно будет создать там нового пользователя и дать ему права на чтение нужных топиков. Если сервер у вас есть, то вы, безусловно, знаете, как это делается.

Если сервера в достижимом окружении нет, то мы установим еще один контейнер с MQTT-сервером Mosquitto внутри для того, чтобы проверить возможности приложения. В этом случае вам придется самостоятельно посылать тестовые данные в топики, для этого можно воспользоваться утилитами командной строки из состава Mosquitto, а лучше – установленным на рабочей станции MQTT-клиентом. Например, для MacOSX есть доступный в App Store клиент MQTT Explorer (http://mqtt-explorer.com), на сайте есть версии и для других платформ.

1. В файл `docker-compose.yml` добавьте еще один сервис:

```
version: '3.3'

services:
  postgres-uitabs:
  ...
  ...

  mosquitto:
    container_name: mosquitto
    image: eclipse-mosquitto
    restart: unless-stopped
    ports:
      - "1883:1883/tcp"
    environment:
      - TZ=Europe/Moscow
    volumes:
      - /opt/mosquitto/config:/mosquitto/config
      - /opt/mosquitto/data:/mosquitto/data
      - /opt/mosquitto/log:/mosquitto/log
    stdin_open: true
    tty: true

volumes:
  uitabs-data:
```

2. Создайте конфигурационный файл для Mosquitto. В качестве заготовки можно скачать файл по умолчанию

```
$  cd /opt/mosquitto/config/
```

```
$  sudo wget
   https://raw.githubusercontent.com/eclipse/mosquitto/master/mosquitto
   .conf
```

3. Отредактируйте конфигурационный файл примерно так, как указано ниже. Можно просто дописать все это в конец файла, а можно раскомментировать соответствующие строки.

```
# Listen on port 1883 on all IPv4 interfaces
listener 1883
socket_domain ipv4
# save the in-memory database to disk
persistence true
persistence_location /mosquitto/data/
# Log to stderr and logfile
log_dest stderr
log_dest file /mosquitto/log/mosquitto.log
# Require authentication
allow_anonymous false
password_file /mosquitto/config/mqttuser
```

4. Запустите докер

```
$  sudo docker compose up -d
```

После запуска docker скачает текущую версию контейнера и запустит его. У вас на локальном порту 1883 есть MQTT-сервер, который вас слушает, но к себе не пустит. Потому что нужна авторизация.

5. Создайте пользователя для доступа к MQTT-серверу. Вместо имени пользователя uitabs можно указать любое, какое вам нравится. Нужно будет придумать и ввести пароль.

```
$  sudo docker exec -it mosquitto mosquitto_passwd -c
   /mosquitto/config/mqttuser uitabs
```

6. В файл `.env` добавьте логин и пароль пользователя Mosquitto.

```
MQTT_USER=uitabs
MQTT_PASSWORD=mymqttpassword
```

7. Перезапустите контейнер с Mosquitto:

```
$  sudo docker compose restart mosquitto
```

4.2 Установка демонстрационного приложения

Тут все проще простого.

Скопируйте дистрибутивный файл `uitabs-demo.app-X.X.X.X.jar` в подходящую директорию (например, `/opt/uitabs`).

Рядом с файлом создайте файл `application.yaml` такого содержания:

```
spring:
  datasource:
    url: jdbc:postgresql://localhost/${POSTGRES_DB}
    username: ${POSTGRES_USER}
    password: ${POSTGRES_PASSWORD}
mqtt:
  server: tcp://localhost:1883
```

```
  username: ${MQTT_USER}
  password: ${MQTT_PASSWORD}
```

Импортируйте переменные из файла `/opt/.env` в текущую сессию:

```
$  set -a
$  source /opt/.env
$  set +a
```

! Не забудьте последнюю команду `set +a` !

Или экспортируйте переменные в шелл другим способом. Или запишите логины и пароли в файл `application.yaml` прямым текстом. Как вам удобнее.

Запустите приложение:

```
$  java -jar /opt/uitabs/uitabs-demo-app-X.X.X.X.jar
```

Можно добавить юнит systemd для автоматического запуска. В файл `/etc/systemd/system/uitabs.service` впишите примерно такое:

```
[Unit]
Description=UITABS Demo application
Requires=docker
After=docker

[Service]
User=<some user>
EnvironmentFile=/opt/.env
ExecStart=/usr/bin/java -jar /opt/uitabs/uitabs-demo-app-X.X.X.X.jar
WorkingDirectory=/opt/uitabs
Restart=on-failure

[Install]
WantedBy=multi-user.target
```

В браузере перейдите по адресу http://localhost:8080/.

Вы должны увидеть страничку с карточками датчиков холодной и горячей воды, влажности и температуры. Карточки будут видны, но значений в них не будет, и статус датчиков будет «Не активен». Потому что приложение надо настраивать.

Можно осмотреть меню и разные странички. Многие ссылки будут отправлять вас на https://google.com. Этого не надо пугаться, просто так в демонстрационном приложении настроено по умолчанию. С помощью настроек системы все это поведение можно переопределить.

4.3 Передача данных в демонстрационное приложение

Как уже было сказано выше, система отображает данные, полученные в сообщениях от MQTT-сервера. Например, у меня дома есть реально работающий MQTT-сервер, на который мои домашние датчики расхода воды, датчики влажности и температуры в комнатах, датчики движения, электрические лампочки и выключатели регулярно отсылают свои состояния.

Каждый датчик публикует свое состояние в отдельном топике – разделе в иерархической структуре данных. Название этих топиков определяются прошивками или настройками каждого датчика и, в общем случае, от нашего желания не зависят.

На нашем тестовом стенде нет реальных показаний датчиков, поэтому нужно их смоделировать. Начнем с определения топиков. Предположим, что датчики потребления воды публикуют свои сообщения в топики `waterius/9315/ch1` и `waterius/9315/ch0` – соответственно, холодная и горячая вода. В каждом сообщении содержится текущее накопленное значение расхода в м^3. Датчик температуры воздуха публикует данные в топик `climate/temperature`, значение текущей замеренной температуры в ^{o}C. Датчик влажности публикует данные в топик `climate/humidity`, значение – текущая замеренная относительная влажность воздуха в процентах.

Обо всем этом нужно рассказать демонстрационному приложению.

Остановите приложение и отредактируйте файл `/opt/uitabs/application.yaml`. Изменения выделены жирным шрифтом:

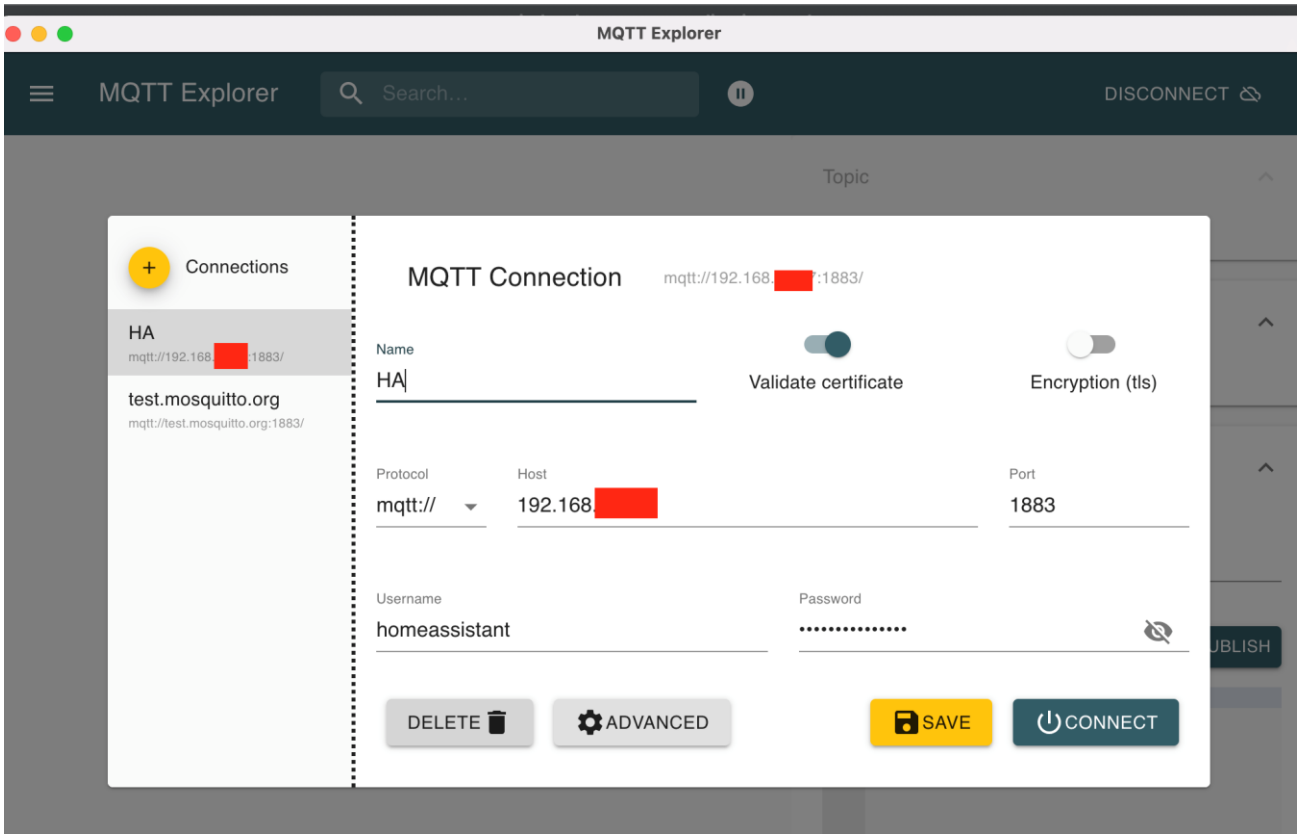
```
spring:
  datasource:
    url: jdbc:postgresql://localhost/${POSTGRES_DB}
    username: ${POSTGRES_USER}
    password: ${POSTGRES_PASSWORD}
mqtt:
  server: tcp://localhost:1883
  username: ${MQTT_USER}
  password: ${MQTT_PASSWORD}
  # Попросим систему подписаться на все сообщения в топиках,
  # расположенных в иерархиях, начинающихся с waterius/ и climate/.
  # О том, как адресуются топики и какой синтаксис можно применить,
  # подробно написано в документации Mosquitto.
  subscribe:
    - "waterius/#"
    - "climate/#"

# Определяем, какой топик что обозначает. Тут мы определили четыре
# отображаемых объекта. Для каждого объекта должен быть определен
# id. Его можно выбирать произвольно, но пока что мы обозначим
# наши объекты именно так, как указано ниже, чтобы сработала
# конфигурация приложения по умолчанию.
# Мы указали, что данные объектов должны быть получены по MQTT, и
# указали имена топиков для каждого. Кроме того, мы указали, что в
# сообщении топика следует ожидать число с плавающей точкой.
# name для объекта выбирается произвольно. Это просто свойство
# объекта. Его можно будет использовать позднее.
dashboards:
  entity:
    - id: waterius_cold_water
      name: Cold water used
      platform: mqtt
      topic: "waterius/9315/ch1"
      type: float
    - id: waterius_hot_water
```

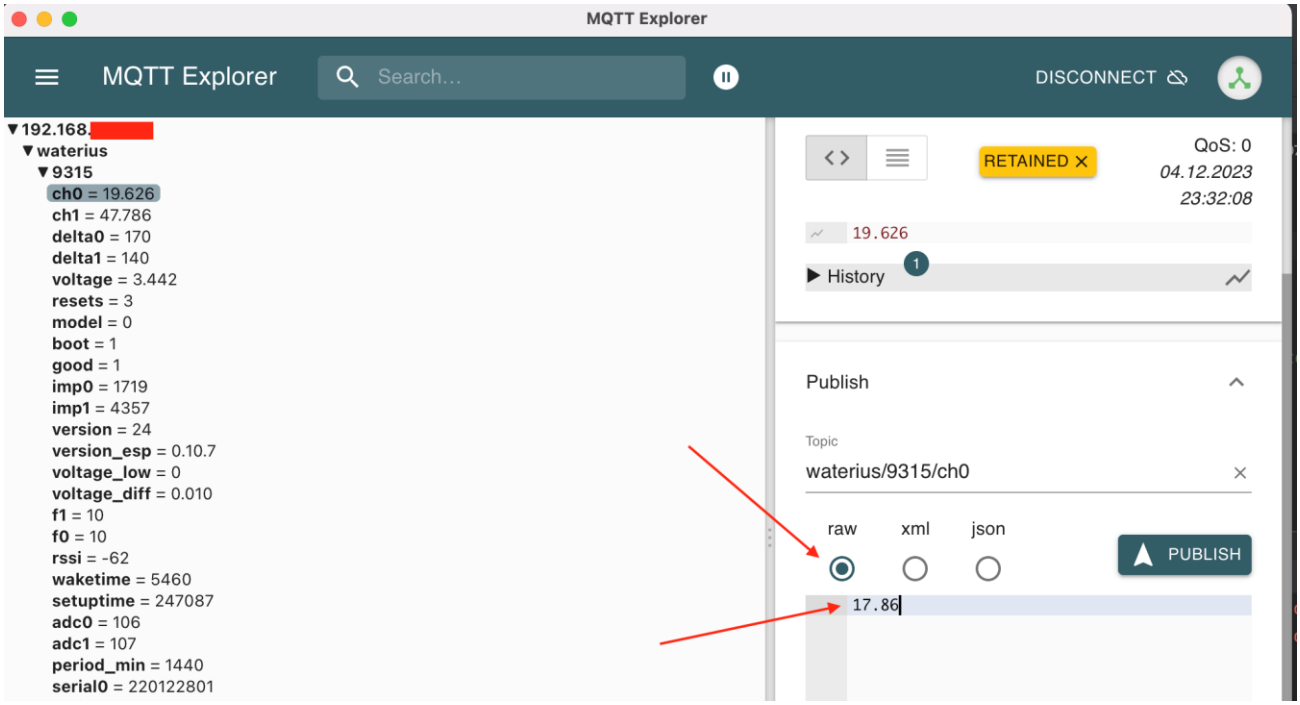
```
    name: Hot water used
    platform: mqtt
    topic: "waterius/9315/ch0"
    type: float
  - id: humidity
    name: Humidity
    platform: mqtt
    topic: "climate/humidity"
    type: float
  - id: temperature
    name: Temperature
    platform: mqtt
    topic: "climate/temperature"
    type: float
```

Запустите приложение и перейдите на его заглавную страницу http://localhost:8080/. Ничего не изменилось, значений датчиков по-прежнему нет, и статус «Не активен». Все логично: наш MQTT-сервер ни от кого еще не получал никаких сообщений.

Запустите MQTT-клиент. Настройте там подключение к нашему тестовому MQTT-серверу, указав там ip-адрес, логин и пароль (тот, который создавали при установке контейнера с Mosquitto). Примерно так, как на рисунке:



Присоединитесь к серверу, найдите в дереве топик `waterius/9315/ch0` и попробуйте отправить туда сообщение с каким-нибудь числом. Если топика в дереве не оказалось (а в первый раз так оно и будет), то ничего страшного, все равно отправляйте сообщение в него, топик будет создан на сервере автоматически.



После этого на странице демонстрационного приложения должно появиться значение счетчика горячей воды, а его состояние должно перейти в «Активен».

Можно посылать сообщения в любые настроенные топики, имитируя деятельность реальных датчиков.

Можно заметить, что после перезапуска приложения все датчики вновь становятся не активными. Это происходит потому, что сообщения в MQTT-топике не сохраняются после их обработки (это нормальное поведение), и когда приложение вновь присоединяется к MQTT-серверу, в топиках оказывается пустота, приложение никаких данных не получает. Чтобы сообщение сохранилось в топике, необходимо установить ему флаг Retained. Такое сообщение будет храниться в топике до тех пор, пока не будет отправлено новое.

5 НАСТРОЙКИ СИСТЕМЫ

Все настройки отображаемых данных, структуры страниц и карточек на аналитической панели, выполняются с помощью файла `application.yaml`. Непосредственно к системе относятся настройки в разделе `dashboards`.

5.1 Шаблонные объекты

В предыдущей главе мы описали четыре информационных объекта, которые отображаются на аналитической панели. Значения этих объектов получены непосредственно из MQTT-сообщения в неизменном виде: что получили, то и отображаем. Если счетчик горячей воды присылает свои показания с точностью до третьего знака после запятой, то именно так они и будут отображены.

Но что, если нам нужно отобразить значение без десятичных знаков? Необходимо округлить значение счетчика. Для этого мы создадим еще один информационный объект.

```
dashboards:
  entity:
    - id: waterius_cold_water
      name: Cold water used
      platform: mqtt
      topic: "waterius/9315/ch1"
      type: float
      # Тут мы пойдем на хитрость: переименуем старый датчик.
      # Ничего хорошего в этом нет, пытливый глаз заметит, что в
      # истории объекта останутся старые значения. Но
      # демонстрационное приложение отображает в своем интерфейсе
      # именно объект с id: waterius_hot_water. Чтобы на данном
      # этапе не менять все настройки, а ограничиться минимумом,
      # так теперь будет называться новый объект с округлением.
    - id: waterius_hot_water_raw
      name: Hot water used
      platform: mqtt
      topic: "waterius/9315/ch0"
      type: float
    - id: waterius_hot_water
      name: Hot water rounded
      platform: template
      value: "{{ waterius_hot_water_raw.state.value | default(0) |
round }}"
      type: float
      import:
        - waterius_hot_water_raw
    - id: humidity
      # Далее все как и было.
      ...
```

Из старого датчика горячей воды, который получал значения из MQTT-сообщений, мы сделали новый, а на старый id повесили другой объект. Он также имеет свойства `id` и `name`, но в свойстве `platform` теперь указано новое значение `template`. Так обозначаются объекты, которые вычисляют свое состояние на основании состояний других объектов или состояний окружающей среды. В свойстве `import` указывается список тех объектов, которые участвуют

в вычислении. В свойстве `value` указано, как вычислять. В качестве значения этого свойства система ожидает увидеть шаблон/скрипт на языке Jinja2 (https://palletsprojects.com/p/jinja/), хорошо известном Python-разработчикам, а также инженерам dev-ops, которые работают с инструментом управления конфигурацией Ansible. В нашем примере скрипт обозначает следующее: «у объекта waterius_hot_water_raw взять текущее состояние, у состояния взять значение, и затем округлить». Вычисление этого скрипта будет производиться всякий раз, когда меняется состояние любого импортированного объекта.

Таким образом, можно определить довольно сложную и обширную систему зависящих друга от друга и автоматически вычисляемых объектов.

Кроме состояния (`state`) в скрипте также можно обращаться к свойствам `name` и `id` объекта. У самого состояния есть два поля: `value` – текущее значение, и `dateTime` – время последнего изменения.

! Обратите внимание на шаблон: "`{{ waterius_hot_water_raw.state.value | default(0) | round }}`". Значение `value` у состояния может оказаться не определено, например, значение с датчика еще не было получено, или получено пустое значение (вот такой вот у нас оказался датчик, с ними это бывает). Фильтр `round` не сможет обработать не определенное значение и возникнет ошибка. Чтобы ее избежать, мы добавили вызов фильтра `default(0)`, который вернет значение 0, если входящее в него значение не определено. Вместо 0 можно поставить любое другое подходящее вам значение.

! Настоятельно рекомендуется использовать эту практику при написании шаблонов!

Кроме однострочных скриптов-шаблонов для вычисления значений можно использовать довольно сложные многострочные конструкции типа таких:

```
      value: >
        {%- set data = namespace(entities=[]) -%}
        {%- for row in temperature|reverse -%}
        {%- if loop.index <= 2 -%}
        {%- set data.entities = data.entities + [[row.time,
row.state.value|string]] -%}
        {%- endif -%}
        {%- endfor -%}
        {{ data.entities | tojson }}
```

Подробная документация по языку Jinja2 расположена по адресу https://jinja.palletsprojects.com/en/3.1.x/templates/#jinja-filters.round. В системе реализованы не все возможности языка, но значительная часть фильтров и проверок работают.

Также при записи шаблонов и скриптов необходимо придерживаться стандарта разметки YAML (https://yaml.org), поскольку именно с помощью него описывается вся конфигурация системы.

5.2 Спецификация полей entity

```
dashboards.entiy := List(<entity>)
entity := <MqttEntity | TemplateEntity>
```

```
MqttEntity := <_ extends BaseEntity>
TemplateEntity := <_ extends BaseEntity>
```

В разделе конфигурации dashboards.entity должен быть указан список объектов entity. В текущей версии доступна настройка entity двух типов: `MqttEntity` и `TemplateEntity`. Оба типа entity содержат как общие поля, так и специфичные для конкретного типа.

Таблица 1. Базовые поля entity.

Поле	Обязательное	Описание
`id`	Да	Строковый идентификатор объекта. Должен быть уникален для каждого объекта в рамках одной конфигурации. По этому идентификатору на объект можно ссылаться в других частях конфигурации.
`name`	Нет	Имя объекта. Задается в человекочитаемом виде, может отображаться в пользовательском интерфейсе для удобства. По умолчанию не задано.
`platform`	Да	Тип entity. Может принимать значение `mqtt` или `template`. Настоятельно рекомендуется указывать тип объекта явно, но если вы забудете это сделать, по умолчанию будет создан объект типа `template`.
`type`	Нет	Тип данных состояния объекта. Поддерживаются следующие типы: `int` – целое число, `float` – число с плавающей точкой, `string` – строка, `json` – составной объект в JSON-формате. По умолчанию тип данных – `string`.

Таблица 2. Дополнительные поля MqttEntity.

Поле	Обязательное	Описание
`topic`	Да	Название MQTT-топика, из которого объект будет получать значения состояния..

Таблица 3. Дополнительные поля TemplateEntity.

Поле	Обязательное	Описание
`value`	Нет	Jinja2 шаблон, который будет вычислять значение состояния объекта. Шаблон может быть как однострочным: `value: "{{ value | default(0) }}"` так и многострочным:

		`value: >`
		
		`{% set val = {"a":"aval", "b":"1"} %}`
		
		`{{ val | tojson }}`
		
		В том числе, можно создать и просто константный объект:
		
		`value: "OK"`
		
		По умолчанию значение параметра не задано, то есть вычисление шаблона всегда будет возвращать значение `undefined`.

`import`	Нет	Список id объектов, которые используются в шаблоне для вычисления состояния. Необходимо указывать данный параметр, если шаблон вычисления основан на свойствах других entity, иначе вычисление будет работать неправильно.

5.3 Пользовательский интерфейс

Структура и вид пользовательского интерфейса, отображаемого приложением, также описывается в файле `application.yaml`. Для этого используется раздел `dashboards.layout`.

```
dashboards:
  entity:
  # Тут описание объектов, см. выше.
  ...

  layout:
  # Тут описывается структура интерфейса

    links:
      - home
      - hot_water
      - cold_water
      - sensors
      - sensors_detail
      - humidity
      - temperature
      - about

    header:
    # Верхний заголовочный блок
    ...

    left_panel:
    # Левая панель меню
    ...
```

```
    page:
    # Описания страниц

      # Страница 1
      - id: aaa
        rows:
          - type: cards
            cards:
              - id: ...
        ...

      # Страница 2
      - id: bbb
        ...
```

Layout в приложении может быть только один. Layout состоит из следующих блоков:

1. Верхний заголовочный блок, который, как правило, будет включать в себя логотип, заголовок приложения и верхнее горизонтальное меню.
2. Левая панель, содержащая в себе раскрывающееся меню.
3. Центральный блок, поочередно отображающий определенные в системе страницы. В каждый момент времени отображается только одна страница, но страниц может быть много. Переходы на страницы осуществляются с помощью ссылок в меню.

Поскольку описание интерфейса производится на языке YAML, можно применять различные его встроенные возможности. Так, например, в `layout` указан дополнительный элемент `links`, содержащий список идентификаторов страниц нашей конфигурации. Ниже мы просто ссылаемся на значения этого списка средствами YAML вместо того, чтобы вводить идентификаторы заново. Таким образом, мы защищаемся от случайных опечаток, которые сложно контролировать. Хотя делать так вовсе не обязательно.

5.3.1 Верхний заголовочный блок

```
dashboards:
  ...

  layout:
    ...
    header:
    # Верхний заголовочный блок
      logo:
        src: "https://cdn-icons-
png.flaticon.com/512/1848/1848669.png"
        height: 50
        width: 50
        mobile_height: 32
        mobile_width: 32
      title:
        value: Dashboard
        size: 30px
      theme:
        type: light
        switch_theme: true
```

```
    menu:
      - id: ${dashboards.layout.links[0]}
        label: Главная
        route: "/"
      - id: ${dashboards.layout.links[7]}
        label: О системе
        type: href
        route: 'https://google.com'
    mobile_menu:
      - id: ${dashboards.layout.links[0]}
        label: Главная
        route: "/"
      - id: ${dashboards.layout.links[7]}
        label: О системе
        type: href
        route: 'https://google.com'

  ...
```

В верхнем заголовочном блоке могут присутствовать следующие разделы:

- `logo` – логотип, отображаемый в левом верхнем углу. Для него указываются параметры:
  - `src` – ссылка (URL) на изображение. Предполагается, что файл изображения располагается на каком-то внешнем (по отношению к приложению) ресурсе.
  - `width` и `height` – ширина и высота изображения в пикселях.
  - `mobile_width` и `mobile_height` - ширина и высота изображения в пикселях при отображении на мобильных устройствах.
- `title` – заголовок, название всей аналитической панели. Отображается вверху, правее логотипа. Параметры:
  - `value` – текст заголовка
  - `size` – размер шрифта заголовка. Допустимы все единицы измерения размеров шрифта, определенные для CSS-элемента `font-size`.
- `menu` и `mobile_menu` – описание меню, отображаемого в заголовке, на большом экране и мобильных устройствах соответственно. Данное меню одноуровневое, в него имеет смысл выносить какие-то ссылки, к которым может понадобиться быстрый доступ. Представляет собой список пунктов меню. Каждый пункт может включать следующие поля:
  - `id` – идентификатор пункта меню, произвольная строка.
  - `label` – текст пункта меню.
  - `type` – тип ссылки. Для указания ссылки на страницу аналитической панели используется тип ссылки `route`, для внешних ссылок используется тип `href`. Если тип ссылки не указан, то по умолчанию используется `route`.
  - `route` – ссылка пункт меню. Для внешней ссылки указывается полный URL перехода, такая ссылка будет открыта в новом окне браузера. Для ссылки на страницу аналитической панели необходимо указывать идентификатор соответствующей страницы (см. ниже) с символом `/` (слэш) впереди, например, `/pageid`.

! Обратите внимание, что для корректной работы на мобильных устройствах необходимо описать `mobile_menu`, даже если оно по структуре полностью совпадает с основным! В противном случае при переходе на мобильную версию меню отображаться не будет.

- `theme` – позволяет задать тему отображения интерфейса по умолчанию, а также указать, будет ли доступно переключение темы. Параметры:
  - `type` – тема по умолчанию, светлая (`light`) или темная (`dark`).
  - `switch_theme` – отображать ли в интерфейсе переключатель тем. Значение `true` или `false`.

5.3.2 Левая панель

```
dashboards:
  ...

  layout:
    ...
    left_panel:
      menu:
        - id: ${dashboards.layout.links[3]}
          label: Общая информация
          route: /${dashboards.layout.links[3]}
        - id: ${dashboards.layout.links[4]}
          label: Датчики
          children:
            - id: ${dashboards.layout.links[1]}
              label: Датчик горячей воды
              route: /${dashboards.layout.links[1]}
            - id: ${dashboards.layout.links[2]}
              label: Датчик холодной воды
              route: /${dashboards.layout.links[2]}
            - id: ${dashboards.layout.links[5]}
              label: Датчик влажности
              route: /${dashboards.layout.links[5]}
            - id: ${dashboards.layout.links[6]}
              label: Датчик температуры
              route: /${dashboards.layout.links[6]}
        - id: any
          label: Внешняя ссылка
          type: href
          route: 'https://google.com'

    ...
```

В левой панели отображается многоуровневое меню, описываемое элементом menu. Данное меню применимо как к полноэкранной, так и к мобильной версии отображения интерфейса. Структура элемента меню аналогична меню верхнего заголовочного блока, описанной ранее в разделе 5.3.1, за исключением дополнительного поля `children`, которое позволяет

присоединить к пункту меню дочернее подменю. При наличии в пункте меню поля `children`, поле `route` указывать не следует.

5.3.3 Страницы

```
dashboards:
  ...

  layout:
    ...
    page:
      - id: ${dashboards.layout.links[0]}
        title: Датчики
        home: true
        rows:
          - type: paragraph
            value: "Текущие значения:"
          - type: cards
            cards:
              ...

      - id: ${dashboards.layout.links[1]}
        title: Горячая вода
        rows:
          - type: cards
            cards:
              - type: rows
                rows:
                  ...
    ...
```

Основная часть конфигурации аналитической панели состоит из описания страниц. Страницы отображаются в центральной части экрана, в каждый момент времени отображается только одна страница. Переходы между страницами осуществляются с помощью меню, настроенных ранее в заголовочном блоке и левой панели.

Все страницы определяются в виде элементов списка поля `dashboards.page`. Для каждой страницы должны быть указаны следующие поля:

- `id` – идентификатор страницы, произвольная строка. Должен быть уникальным для каждой страницы в конфигурации. Именно по этому идентификатору производится переход на страницу из пункта меню.
- `title` – Текстовый заголовок страницы.
- `rows` – список строк, отображаемых на странице (см. ниже).

Для одной из страниц также можно определить поле `home` (значения `true` или `false`). Страница с `home: true` будет отображаться на панели по умолчанию при первом запуске. Если не задавать поле `home` ни для одной страницы, то страницей по умолчанию будет первая страница, определенная в конфигурации.

5.3.3.1 Строки страниц

Страница состоит из строк. Строки могут быть трех типов:

- `paragraph` – для отображения в строке простого текстового фрагмента. Строго говоря, любого фрагмента, который разрешено отображать внутри HTML-тега `<p>`. При большом количестве текста он будет вытягиваться горизонтально на всю доступную ширину экрана.
- `html` – то же самое, но для отображения произвольного фрагмента HTML.
- `cards` – для отображения сетки прямоугольных карточек, которые определяются в конфигурации строки. Сетка формируется динамически в зависимости от ширины содержимого карточек с помощью CSS Grid Layout.

Каждая строка определяется следующими полями:

- `type` – тип строки.
- `value` – для строк типов `paragraph` и `html` в этом поле указывается фрагмент текста или HTML-разметки для отображения.

! Внимание! В текущей версии системы в поле `value` для строк не поддерживается вычисление Jinja2-выражений! Следует указывать только статичные фрагменты.

- `cards` – для строк типа `cards` в этом поле определяется список отображаемых карточек (см. ниже).

5.3.4 Карточки

Карточка – прямоугольный элемент, отображающий, как правило, какое-то небольшое количество информации, касающееся одного или нескольких объектов entity. Автоматически выравниваемая сетка из карточек формирует внешнее представление аналитической панели.

В каждой строке страницы можно поместить произвольное количество карточек, при этом карточки каждой строки будут формировать свою прямоугольную сетку. В пределах сетки карточки отображаются в порядке их определения в конфигурации, слева направо а затем сверху вниз.

В системе поддерживается несколько типов карточек, обеспечивающих привлекательное и компактное отображение разнообразной информации. В текущей версии поддерживаются следующие типы:

- `row_template` – простая карточка, в которой отображается текстовая информация. Текст может быть как статический, так и вычисляемый с помощью Jinja2-шаблона.

! Название этого типа не удачное, и будет вносить определенную путаницу ниже, при описании полей карточек. Но в текущей версии пока что так.

- `html` – простая карточка, в которой отображается фрагмент HTML. Фрагмент может быть как статический, так и вычисляемый с помощью Jinja2-шаблона.
- `status` – карточка для отображения статуса активности объекта. По значению заданного таймаута и времени последнего обновления состояния объекта отображает, «активен» объект или нет.

- `history` – отображение истории изменения состояния объекта в виде таблицы. В общем случае пригодна для отображения любой табличной информации.
- `gauge` – отображение числового объекта в виде «спидометра».
- `charts` – отображение серии числовых значений в виде линейного графика. Как правило, используется для графического отображения истории состояний объекта, например, можно нарисовать график расхода воды.
- `rows` – композитная карточка, которая сама состоит из строк, которые могут содержать другие карточки.

Определение каждой карточки содержит общие поля:

- `id` – идентификатор карточки, произвольная строка. Крайне желательно, чтобы идентификаторы карточек были уникальны, по крайней мере, в рамках одной страницы панели.
- `type` – тип карточки.
- `title` – заголовок карточки.

Карточки всех типов, кроме типа `rows`, должны содержать поля для описания отображаемых данных:

- `row_template` – Jinja2-шаблон, который будет вычислять значения для отображения на карточке. Для каждого типа карточки шаблон должен возвращать данные определенного типа (см. ниже в описаниях отдельных типов карточек). Шаблон может быть как однострочным:

  ```
  row_template: "{{ value | default(0) }}"
  ```

  так и многострочным:

  ```
  row_template: >
      {% set val = {"a":"aval", "b":"1"} %}
      {{ val | tojson }}
  ```

  В том числе, можно создать и просто константный объект:

  ```
  value: "OK"
  ```

! Я предупреждал, что может возникнуть путаница. Поле `row_template` должно быть определено у карточек любого типа, а не только типа `row_template`.

- `entity_id` – список идентификаторов (`id`) объектов, которые используются в шаблоне для вычисления состояния. Необходимо указывать данный параметр, если шаблон вычисления основан на свойствах других entity, иначе вычисление будет работать неправильно.

Кроме этого, каждый тип карточки может содержать в описании дополнительные поля, которые уточняют способ отображения информации.

5.3.4.1 Карточка row_template

Карточки типа `row_template` не требуют дополнительной настройки.

5.3.4.2 Карточка html

Карточки типа `html` не требуют дополнительной настройки.

5.3.4.3 Карточка status

Для карточки типа `status` необходимо определить дополнительное поле `minutes`. В нем должно быть указано целочисленное значение таймаута неактивности объекта в минутах. Если состояние объекта не изменялось более, чем указанное значение минут, то статус объекта будет отображен как не активный.

В поле `row_template` можно задать выражение, генерирующее дополнительный пояснительный текст, который будет отображен на карточке вместе со статусом.

Если для вычисления шаблона в `row_template` используются несколько объектов, то их идентификаторы необходимо перечислить в поле `entity_id`. При этом карточка будет отображать статус первого объекта, а не всех перечисленных.

Пример определения карточки status:

```
    - id: status_hot_waters
      title: Статус счетчика горячей воды
      type: status
      minutes: 5
      entity_id:
        - waterius_hot_water
      row_template: "последнее значение передано: {{
waterius_hot_water.state.dateTime }}"
```

5.3.4.4 Карточка history

Карточка типа `history` позволяет отображать данные в табличном виде. Для этого шаблон row_template должен возвращать данные в виде JSON-массива следующей структуры:

```
[[row1_col1, row1_col2, row3_col3, ...], [row2_col1, row2_col2,
row2_col3, ...],  ...]
```

Количество строк и колонок не ограничивается.

Если шаблон вычисляется на основании истории состояний объектов, то их идентификаторы необходимо перечислить в поле `entity_id`. В этом случае в шаблоне обращение к идентификатору объекта будет возвращать массив предыдущих состояний. Эти состояния будут отсортированы по дате их фиксации в системе от наиболее новых к более старым.

В дополнительном поле `headers` для карточки задается список заголовков столбцов. Количество заголовков должно соответствовать количеству элементов в строке JSON-массива.

Пример определение карточки `history`:

```
    - id: history
      type: history
      title: Последние 5 показаний
      entity_id:
        - waterius_hot_water
      row-template: >
```

```
        {%- set data = namespace(entities=[]) -%}
        {%- for row in waterius_hot_water -%}
        {%- if loop.index <= 5 -%}
        {%- set data.entities = data.entities + [[row.time,
row.state.value|string]] -%}
        {%- endif -%}
        {%- endfor -%}
        {{ data.entities | tojson }}
      headers:
        - Дата и время
        - Значение
```

5.3.4.5 Карточка gauge

Карточка типа gauge отображает числовое значение в виде «спидометра». Значение может вычисляться на основании состояний одного или нескольких объектов, при этом шаблон в поле `row_template` должен возвращать число. Идентификаторы объектов, участвующих в вычислении, должны быть перечислены в поле `entity_id`.

Чтобы определить внешний вид «спидометра», в карточке нужно заполнить поле `gauge`.

- `gauge.max_value` – максимальное значение, которое может быть отображено на «спидометре». На этом значении стрелка будет «ложиться».
- `gauge.unit` – можно указать строчку с единицами измерения величины, отображаемой на «спидометре». Например, км/ч или $м^3$.
- `gauge.ticks` – список значений, для которых на «спидометре» будут нарисованы засечки.
- `gauge.sub_args` – список пар значений, определяющих цвета подложки «спидометра» для диапазонов отображаемых значений. Например, диапазон скоростей до 60 км/ч может отображаться на «спидометре» зеленым цветом, от 60 км/ч до 130 км/ч – желтым, а скорости выше 130 км/ч - красным. Для каждой границы изменения цвета указывается пара полей:
  - `gauge.sub_args.limit` – граничное значение величины.
  - `gauge.sub_args.color` – цвет подложки «спидометра» ДО граничного значения, в формате определения цвета HTML/CSS.

  В последнем элементе списка можно указать два поля `color`. Второе поле будет определять цвет подложки ПОСЛЕ граничного значения и до конца шкалы.

  Для каждой границы изменения цвета подложки также будут нарисованы засечки.

Пример определения карточки `gauge`:

```
    - id: gauge
      type: gauge
      entity_id:
        - waterius_hot_water
      row_template: "{{ waterius_hot_water.state.value|default(0)
}}"
      gauge:
        sub_args:
          - limit: 50
```

```
            color: '#c01f1f'
        ticks:
          - 10
          - 20
          - 30
          - 40
        max_value: 50
        unit: м³
```

5.3.4.6 Карточка charts

Карточка типа charts позволяет отобразить данные в виде линейного графика. На одном графике можно отобразить один или несколько рядов данных разными линиями. Для этого шаблон row_template должен возвращать данные в виде JSON-массива следующей структуры:

```
[{x: x0, line1: y0_1, line2: y0_2}, {x: x1, line1: y1_1, line2:
y1_2}, {x: x2, line1: y2_1, line2: y2_2}, ...]
```

Каждый элемент массива описывает точки нескольких рядов данных: ряд значений оси абсцисс (x) и один или несколько рядов ординат для линий графика, в данном случае, `line1` и `line2`. Можно использовать любое количество линий и количество точек графика (в пределах разумного). Идентификаторы для линий (`line1` и `line2` в данном случае) тоже можно выбирать произвольно.

Если шаблон вычисляется на основании истории состояний объектов, то их идентификаторы необходимо перечислить в поле `entity_id`. В этом случае в шаблоне обращение к идентификатору объекта будет возвращать массив предыдущих состояний этого объекта. Эти состояния будут отсортированы по дате их фиксации в системе от наиболее новых к более старым.

Чтобы задать внешний вид графика можно использовать следующие дополнительные поля настройки карточки:

width – ширина графика в пикселях.

height – высота графика в пикселях.

line_names – наименование рядов данных для обозначения осей и составления легенды для графика. Представляет собой список отображений идентификаторов рядов в человеко-читаемые названия. В поле line_names.key указывается идентификатор ряда, в поле line_names.value – название.

Пример определения карточки `charts`:

```
    - id: charts
      type: charts
      height: 600
      width: 1000
      entity_id:
        - test_json_template
      row-template: >
        {%- set data = namespace(entities=[]) -%}
        {%- for row in test_json_template|reverse -%}
```

```
        {%- set data.entities = data.entities + [{x: row.time,
hotWater: row.state.value.hotWater, coldWater:
row.state.value.coldWater}] -%}
        {%- endfor -%}
        {{ data.entities | tojson }}
      line_names:
        - key: hotWater
          value: горячая вода
        - key: coldWater
          value: холодная вода
        - key: x
          value: дата и время
```

! Обратите внимание, что в шаблоне список истории состояний реверсируется! Это делается для того, чтобы на графике состояния отображались от старых (слева) к более новым (справа). Система возвращает в шаблон список состояний в обратном порядке!

! Также обратите внимание, что здесь для отображения двух графиков одновременно без пропуска точек используется шаблонный объект, который фиксирует в своей истории состояний значения обоих счетчиков воды одновременно при изменении значения любого из них. Вот определение такого объекта:

```
dashboards:
  entity:
    ...

    - id: test_json_template
      name: Test JSON template
      platform: template
      value: >
        {% set val =
{'coldWater':waterius_cold_water.state.value|default(0),
'hotWater':waterius_hot_water.state.value|default(0)} %}
        {{ val | tojson }}
      type: json
      import:
        - waterius_cold_water
        - waterius_hot_water
```

5.3.4.7 Карточка rows

Карточка типа rows представляет собой «страницу в миниатюре». Ее структура полностью соответствует странице: карточка состоит из строк, в которых могут быть другие карточки (если строка имеет тип cards) или фрагменты текста/HTML (типы paragraph/html).

Строки для карточки указываются списком в поле rows. Поля row_template и entity_id к карточкам этого типа не применимы.

Пример определения карточки `rows`:

```
    - id: both_water_status
```

```
      type: rows
      rows:
        - type: cards
          cards:
            - id: status_hot_waters
              title: Статус датчика влажности
              type: status
              minutes: 5
              entity_id:
                - humidity
              row_template: "последнее значение передано: {{
humidity.state.dateTime }}"
            - id: status_hot_waters
              title: Статус датчика температуры
              type: status
              minutes: 5
              entity_id:
                - temperature
              row_template: "последнее значение передано: {{
temperature.state.dateTime }}"
```

5.3.5 Настройки системы

В разделе `dashboards.settings` можно настроить аспекты поведения системы.

Для того, чтобы не допустить бесконтрольного роста базы данных состояний, при обновлении состояния объекта система производит очистку его устаревших состояний. По умолчанию хранятся состояния за 48 часов, при каждом обновлении объекта более старые будут удалены, но при этом если в БД осталось единственное состояние старше 48 часов, то оно будет оставлено, чтобы иметь возможность видеть, когда произошло последнее обновление. Срок хранения состояний можно изменить, указав в конфигурации параметр `dashboards.settings.history_retention`. Значение данного параметра – это `java.time.Duration`, оно может быть указано в формате ISO 8601 (https://www.digi.com/resources/documentation/digidocs//90001488-13/reference/r_iso_8601_duration_format.htm) или в упрощенном формате `<число>единицы` (например, 48h), где единицами могут быть следующие:

- `ns`: наносекунды
- `us`: микросекунды
- `ms`: миллисекунды
- `s`: секунды
- `m`: минуты
- `h`: часы
- `d`: дни

При загрузке истории состояний из БД система ограничивает количество загруженных состояний по двум параметрам одновременно: не старше определенного возраста и не более установленного количества состояний. По умолчанию возраст определен 48 часов, а количество – 200 состояний. При желании эти ограничения можно переопределить в параметрах `dashboards.settings.history_retrieval_duration` и `dashboards.settings.history_retrieval_length`. Первый параметр –

`java.time.Duration`, настраивается так же, как и `history_retention`, второй – целое число.

Настройки по умолчанию:

```
dashboards:
  settings:
    history_retention: 48h
    history_retrieval_length: 200
    history_retrieval_duration: 48h
```

6 ИСПОЛЬЗОВАНИЕ КОМПОНЕНТОВ СИСТЕМЫ ДЛЯ ВСТРАИВАНИЯ В СВОИ ПРИЛОЖЕНИЯ

Система позволяет встроить себя в периметр внешней информационной системы. Это можно делать различными способами.

6.1 Интеграция как сервис

Самый простой и очевидный путь – запустить демонстрационное приложение как отдельный сервис в информационной системе, предварительно настроив его под свои нужды с помощью конфигурации в `application.yaml`. Все, что нужно для работы приложения – это MQTT-протокол на входе и HTTP/HTTPS на выходе. Данные для отображения из вашей системы будут поступать через MQTT-сервер. Пользовательский интерфейс будет доступен в любом удобном месте через Nginx reverse proxy.

Такой подход обладает многими очевидными преимуществами. По сути, все, что требуется от интегратора – реализовать своевременную отправку данных по хорошо известному и стандартизированному протоколу MQTT. В современных условиях, когда архитектура многих информационных систем вообще целиком строится на передаче сообщений (messaging), эта задача выглядит достаточно простой и недорогой. Нет необходимости изучать проприетарные API, все можно сделать с помощью общедоступных библиотек, которые существуют для любых платформ. MQTT-сервер также можно выбрать себе по вкусу и по карману, а в случае необходимости – оперативно его заменить. Настройка отказоустойчивой конфигурации с распределением нагрузки не представляет собой никакой проблемы ни для HTTP-интерфейса приложения, ни для базы данных Postgres, в которой приложение хранит историю состояний объектов. MQTT-серверы с возможностью кластеризации также доступны на рынке, в том числе и бесплатные.

Единственным ограничением текущей версии системы при таком подходе является невозможность задать в Nginx префикс для URL-ов, на которых работает приложение. Данные в пользовательский интерфейс поступают путем запросов к фиксированным URL `/uitabs/v1/<method>`, поэтому добавлять впереди префикс (например, `/mydashboard/uitabs/v1`) нельзя. Но вероятность того, что префикс `/uitabs` уже занят чем-то полезным на вашем веб-сервере, согласитесь, не очень высока.

6.2 Интеграция в Spring Boot приложение

Этот способ более трудоемкий, но позволит вам полностью интегрировать аналитические панели в ваше приложение. Вы сможете настроить права доступа к панели с помощью Spring Security, хранить состояния объектов в удобном вам месте и т.п.

В комплект поставки системы включения две Java- библиотеки: `uitabs-core-x.x.x.jar` и `uitabs-backend-x.x.x.jar` для интеграции в приложение Spring Boot, а также npm-библиотека `uitabs-ui-x.x.x.tgz` для встраивания в React-приложения пользовательского интерфейса. Библиотекой `uitabs-core-x.x.x.jar` вряд ли вы будете пользоваться напрямую, но от нее зависит библиотека `uitabs-backend-x.x.x.jar`.

6.2.1 Базовая интеграция в Spring Boot

Поскольку система имеет коммерческую лицензию, библиотеки не распространяются через Maven Central repository или иные репозитории пакетов. Это значит, что вам нужно будет предварительно загрузить их в свой локальный репозиторий. На крайний случай – установить их себе в `$HOME/.m2/` с помощью команды вида:

```
$ mvn install:install-file \
   -Dfile=uitabs-core-x.x.x.jar \
   -DgroupId=ru.itlabpro.uitabs \
   -DartifactId=uitabs-core \
   -Dversion=x.x.x \
   -Dpackaging=jar \
   -DpomFile=pom.xml
```

POM-файлы упакованы в самих библиотеках в `META-INF/maven/ru.itlabpro.uitabs/uitabs-<lib>`, их надо оттуда достать.

Затем убедитесь, что ваше приложение использует Spring Boot, то есть ваш POM-файл наследуется от

```
   <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.1.3</version>
        <relativePath /> <!-- lookup parent from repository -->
   </parent>
```

Крайне желательна версия Spring Boot 3.1 или выше.

После этого дело сводится к добавлению в ваш POM-файл зависимостей:

```
        <dependency>
             <groupId>org.springframework.boot</groupId>
             <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>

        <!-- Можно использовать и другую БД, -->
        <!-- которая у вас уже есть. -->
        <dependency>
             <groupId>org.postgresql</groupId>
             <artifactId>postgresql</artifactId>
        </dependency>

        <dependency>
             <groupId>org.liquibase</groupId>
             <artifactId>liquibase-core</artifactId>
        </dependency>

        <dependency>
             <groupId>ru.itlabpro.uitabs</groupId>
             <artifactId>uitabs-backend</artifactId>
             <version>x.x.x</version>
        </dependency>


        <!-- Можно и не добавлять, но тогда будут работать -->
```

```
        <!-- только шаблонные объекты. -->
        <dependency>
             <groupId>org.eclipse.paho</groupId>
             <artifactId>org.eclipse.paho.mqttv5.client</artifactId>
             <version>1.2.5</version>
        </dependency>
```

Библиотека использует механизм Spring Boot Autoconfiguration, поэтому все необходимые бины должны добавиться в контекст сами.

Для создания таблиц и индексов БД библиотека использует Liquibase, следовательно, вам нужно добавить в ваш changelog `src/main/resources/db/changelog-master.xml` строку с включением библиотечного changelog'а:

```
    <include file="classpath:/db/uitabs/changelog-master.xml"/>
```

Добавьте в ваш `application.yaml` разделы `mqtt` и `dashboards` с необходимой вам конфигурацией.

В результате вы получите, фактически, аналог демонстрационного приложения, только со своей конфигурацией. По крайней мере, демонстрационное приложение устроено именно так.

6.2.2 Интеграция в приложение React

Для пользовательского интерфейса панели используется библиотека компонентов `uitabs-ui-x.x.x.tgz`. Давайте предположим, что в вашем приложении пользовательский интерфейс тоже строится на базе React 17, в этом случае вы сможете очень легко интегрировать компоненты панели. Положите библиотеку, скажем, в директорию `lib` внутри вашего React-приложения и добавьте в `package.json` зависимость:

```
    "dependencies": {
        "uitabs-ui": "file:lib/uitabs-ui-x.x.x.tgz",
        …
    }
```

В подходящем месте импортируйте компонент Dashboard из библиотеки и вставьте его в дерево компонентов. Например, в виде корневого элемента:

```
/* File: index.tsx */

import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import 'normalize.css';
import {Dashboard} from 'uitabs-ui'
import queryString from "querystring";
import moment from "moment";
import axios from "axios";

axios.defaults.paramsSerializer =  (params) =>
queryString.stringify(params);

ReactDOM.render(<Dashboard/>, document.getElementById('root'));
```

В таблице стилей `index.css` у нас лежит немного украшений:

```
/* File: index.css */
```

```
::-webkit-scrollbar {
    width: 0;
}

html {
    scrollbar-width: none;
    -ms-overflow-style: none;
}

a {
    font-size: 14px;
}

table {
    width: 100%;
    border: none;
}
table thead th {
    text-align: left;
    border: none;
    padding: 10px 15px;
    font-size: 14px;
    font-weight: normal;
}
table tbody td {
    text-align: left;
    padding: 10px 15px;
    font-size: 14px;
    vertical-align: top;
}
table thead tr th:first-child, table tbody tr td:first-child {
    border-left: none;
}
table thead tr th:last-child, table tbody tr td:last-child {
    border-right: none;
}
```

К сожалению, если вы используете React 18, то фокус не получится, в текущей версии библиотека с ним не совместима.

7 ПРИЛОЖЕНИЕ 1. КОНФИГУРАЦИЯ ДЕМОНСТРАЦИОННОГО ПРИЛОЖЕНИЯ

Ниже приведена полная конфигурация демонстрационного приложения аналитической панели, поставляемая по умолчанию в комплекте поставки.

```
spring:
  datasource:
    url: jdbc:postgresql://localhost/dashboard
    username: admin
    password: admin
    hikari:
      connection-test-query: SELECT 1
      maximum-pool-size: 50
  liquibase:
    change-log: "classpath:/db/changelog-master.xml"
    contexts: "!test"

---
mqtt:
  server: tcp://192.168.84.17:1883
  username: homeassistant
  password: ${DASHBOARD_MQTT_PASSWORD}
  subscribe:
    #    - "zigbee2mqtt/#"
    - "waterius/#"

dashboards:
  settings:
    history_retention: 48h
    history_retrieval_length: 200
    history_retrieval_duration: 48h
  entity:
    - id: waterius_cold_water
      name: Cold water used
      platform: mqtt
      topic: "waterius/9315/ch1"
      type: float
    - id: waterius_hot_water
      name: Hot water used
      platform: mqtt
      topic: "waterius/9315/ch0"
      type: float
    - id: humidity
      name: Humidity
      platform: mqtt
      topic: "waterius/9315/ch3"
      type: float
    - id: temperature
      name: Temperature
      platform: mqtt
      topic: "waterius/9315/ch2"
```

```
      type: float
    - id: room_home_lobby
      name: Lobby at home
      platform: template
      value: Lobby
    - id: template_test
      name: Testing template
      platform: template
      value: "{{ waterius_cold_water.state.value * 10 }}"
      type: float
      import:
        - waterius_cold_water
    - id: test_json_template
      name: Test JSON template
      platform: template
      value: >
        {% set val =
{'coldWater':waterius_cold_water.state.value|default(0),
'hotWater':waterius_hot_water.state.value|default(0)} %}
        {{ val | tojson }}
      type: json
      import:
        - waterius_cold_water
        - waterius_hot_water

  layout:
    links:
      - home
      - hot_water
      - cold_water
      - sensors
      - sensors_detail
      - humidity
      - temperature
      - about
    header:
      logo:
        src: "https://cdn-icons-
png.flaticon.com/512/1848/1848669.png"
        height: 50
        width: 50
        mobile_height: 32
        mobile_width: 32
      title:
        value: Dashboard
        size: 30px
      theme:
        type: light
        switch_theme: true
      menu:
        - id: ${dashboards.layout.links[0]}
          label: Главная
          route: "/"
        - id: ${dashboards.layout.links[7]}
```

```
        label: О системе
        type: href
        route: 'https://google.com'
    mobile_menu:
      - id: ${dashboards.layout.links[0]}
        label: Главная
        route: "/"
      - id: ${dashboards.layout.links[7]}
        label: О системе
        type: href
        route: 'https://google.com'
  left_panel:
    menu:
      - id: ${dashboards.layout.links[3]}
        label: Общая информация
        route: /${dashboards.layout.links[3]}
      - id: ${dashboards.layout.links[4]}
        label: Датчики
        children:
          - id: ${dashboards.layout.links[1]}
            label: Датчик горячей воды
            route: /${dashboards.layout.links[1]}
          - id: ${dashboards.layout.links[2]}
            label: Датчик холодной воды
            route: /${dashboards.layout.links[2]}
          - id: ${dashboards.layout.links[5]}
            label: Датчик влажности
            route: /${dashboards.layout.links[5]}
          - id: ${dashboards.layout.links[6]}
            label: Датчик температуры
            route: /${dashboards.layout.links[6]}
      - id: any
        label: Внешняя ссылка
        type: href
        route: 'https://google.com'
  page:
    - id: ${dashboards.layout.links[0]}
      title: Датчики
      rows:
        - type: paragraph
          value: "Текущие значения:"
        - type: cards
          cards:
            - id: hot_and_cold_water
              title: Датчики воды
              type: html
              row_template: "<table>
              <thead>
              <tr>
              <th>Горячая вода</th>
              <th>Холодная вода</th>
              </tr>
              </thead>
              <tbody>
```

```
                <tr>
                <td>{{ waterius_hot_water.state.value|default(0) }}
м³</td>
                <td>{{ waterius_cold_water.state.value|default(0) }}
м³</td>
                </tr>
                </tbody>
                </table>"
                entity_id:
                  - waterius_hot_water
                  - waterius_cold_water
              - id: sensors
                title: Текущие значения датчиков
                type: html
                row_template: "<table>
                <thead>
                <tr>
                <th>Датчик</th>
                <th>Значение</th>
                </tr>
                </thead>
                <tbody>
                <tr>
                <td>Горячая вода</td>
                <td>{{ waterius_hot_water.state.value|default(0) }}
м³</td>
                </tr>
                <tr>
                <td>Холодная вода</td>
                <td>{{ waterius_cold_water.state.value|default(0) }}
м³</td>
                </tr>
                <tr>
                <td>Датчик влажности</td>
                <td>{{ humidity.state.value|default(0) }} %</td>
                </tr>
                <tr>
                <td>Датчик температуры</td>
                <td>{{ temperature.state.value|default(0) }} °C</td>
                </tr>
                </tbody>
                </table>"
                entity_id:
                  - waterius_hot_water
                  - waterius_cold_water
                  - humidity
                  - temperature
              - type: rows
                rows:
                - type: cards
                  cards:
                    - id: status_hot_waters
                      title: Статус счетчика горячей воды
                      type: status
```

```
                      minutes: 5
                      entity_id:
                        - waterius_hot_water
                      row_template: "последнее значение передано: {{
waterius_hot_water.state.dateTime | default('Неизвестно когда') }}"
                    - id: status_hot_waters
                      title: Статус счетчика холодной воды
                      type: status
                      minutes: 5
                      entity_id:
                        - waterius_cold_water
                      row_template: "последнее значение передано: {{
waterius_cold_water.state.dateTime | default('Неизвестно когда') }}"
              - type: rows
                rows:
                - type: cards
                  cards:
                    - id: status_hot_waters
                      title: Статус датчика влажности
                      type: status
                      minutes: 5
                      entity_id:
                        - humidity
                      row_template: "последнее значение передано: {{
humidity.state.dateTime | default('Неизвестно когда')}}"
                    - id: status_hot_waters
                      title: Статус датчика температуры
                      type: status
                      minutes: 5
                      entity_id:
                        - temperature
                      row_template: "последнее значение передано: {{
temperature.state.dateTime | default('Неизвестно когда')}}"
          - type: html
            value: "<p>посмотреть инструкцию: <a
href=\"https://google.com\">ссылка</a></p>"
      - id: ${dashboards.layout.links[1]}
        title: Горячая вода
        rows:
          - type: cards
            cards:
              - type: rows
                rows:
                    - type: cards
                      cards:
                        - id: ${dashboards.layout.links[1]}
                          type: row_template
                          entity_id:
                            - waterius_hot_water
                          row_template: "Текущее значение: {{
waterius_hot_water.state.value|default(0) }} м³"
                        - id: status_hot_waters
                          title: Статус счетчика горячей воды
                          type: status
```

```
                          minutes: 5
                          entity_id:
                            - waterius_hot_water
                          row_template: "последнее значение
передано: {{ waterius_hot_water.state.dateTime | default('Неизвестно
когда')}}"
                        - id: gauge
                          type: gauge
                          entity_id:
                            - waterius_hot_water
                          row_template: "{{
waterius_hot_water.state.value|default(0) }}"
                          gauge:
                            sub_args:
                              - limit: 50
                                color: '#c01f1f'
                            ticks:
                              - 10
                              - 20
                              - 30
                              - 40
                            max_value: 50
                            unit: м³
              - id: history
                type: history
                title: Последние 5 показаний
                entity_id:
                  - waterius_hot_water
                row-template: >
                  {%- set data = namespace(entities=[]) -%}
                  {%- for row in waterius_hot_water -%}
                  {%- if loop.index <= 5 -%}
                  {%- set data.entities = data.entities +
[[row.time, row.state.value|string]] -%}
                  {%- endif -%}
                  {%- endfor -%}
                  {{ data.entities | tojson }}
                headers:
                  - Дата и время
                  - Значение, м³
              - id: charts
                type: charts
                height: 400
                width: 500
                entity_id:
                  - waterius_hot_water
                row-template: >
                  {%- set data = namespace(entities=[]) -%}
                  {%- for row in waterius_hot_water|reverse -%}
                  {%- set data.entities = data.entities + [{x:
row.time, y: row.state.value}] -%}
                  {%- endfor -%}
                  {{ data.entities | tojson }}
                line_names:
```

```
                  - key: y
                    value: горячая вода
                  - key: x
                    value: дата и время

      - id: ${dashboards.layout.links[2]}
        title: Холодная вода
        rows:
          - type: cards
            cards:
              - id: ${dashboards.layout.links[2]}
                type: row_template
                entity_id:
                  - waterius_cold_water
                row_template: "Текущее значение: {{
waterius_cold_water.state.value|default(0) }} м³"
              - id: status_cold_waters
                title: Статус счетчика горячей воды
                type: status
                minutes: 5
                entity_id:
                  - waterius_cold_water
                row_template: "последнее значение передано: {{
waterius_cold_water.state.dateTime | default('Неизвестно когда')}}"
              - id: gauge
                type: gauge
                entity_id:
                  - waterius_cold_water
                row_template: "{{
waterius_cold_water.state.value|default(0) }}"
                gauge:
                  sub_args:
                    - limit: 50
                      color: 'blue'
                  ticks:
                    - 10
                    - 20
                    - 30
                    - 40
                    - 50
                    - 60
                  max_value: 70
                  unit: м³
          - type: cards
            cards:
              - id: charts
                type: charts
                height: 400
                width: 1005
                entity_id:
                  - waterius_cold_water
                row-template: >
                  {%- set data = namespace(entities=[]) -%}
                  {%- for row in waterius_cold_water|reverse -%}
```

```
                  {%- set data.entities = data.entities + [{x:
row.time, y: row.state.value}] -%}
                  {%- endfor -%}
                  {{ data.entities | tojson }}
                line_names:
                  - key: y
                    value: холодная вода
                  - key: x
                    value: дата и время

      - id: ${dashboards.layout.links[3]}
        title: Общая информация
        rows:
          - type: cards
            cards:
              - id: charts
                type: charts
                height: 600
                width: 1000
                entity_id:
                  - test_json_template
                row-template: >
                  {%- set data = namespace(entities=[]) -%}
                  {%- for row in test_json_template|reverse -%}
                  {%- set data.entities = data.entities + [{x:
row.time, hotWater: row.state.value.hotWater, coldWater:
row.state.value.coldWater}] -%}
                  {%- endfor -%}
                  {{ data.entities | tojson }}
                line_names:
                  - key: hotWater
                    value: горячая вода
                  - key: coldWater
                    value: холодная вода
                  - key: x
                    value: дата и время

      - id: ${dashboards.layout.links[6]}
        title: Температура
        rows:
          - type: cards
            cards:
              - id: gauge
                type: gauge
                entity_id:
                  - temperature
                row_template: "{{ temperature.state.value|default(0)
}}"
                gauge:
                  type: semicircle
                  sub_args:
                    - limit: 15
                      color: '#EA4228'
                    - limit: 17
```

```
                    color: '#F5CD19'
                  - limit: 28
                    color: '#5BE12C'
                  - limit: 30
                    color: '#F5CD19'
                  - color: '#EA4228'
                ticks:
                  - 13
                  - 22
                  - 32
                min_value: 10
                max_value: 35
                unit: °C
              - id: history
                type: history
                title: Последние 2 показания
                entity_id:
                  - temperature
                row-template: >
                  {%- set data = namespace(entities=[]) -%}
                  {%- for row in temperature -%}
                  {%- if loop.index <= 2 -%}
                  {%- set data.entities = data.entities +
[[row.time, row.state.value|string]] -%}
                  {%- endif -%}
                  {%- endfor -%}
                  {{ data.entities | tojson }}
                headers:
                  - Дата и время
                  - Температура, °C
      - id: ${dashboards.layout.links[5]}
        title: Влажность
        rows:
          - type: cards
            cards:
              - id: gauge
                type: gauge
                entity_id:
                  - humidity
                row_template: "{{ humidity.state.value|default(0)
}}"
                gauge:
                  type: radial
                  sub_args:
                    - limit: 30
                      color: '#F5CD19'
                    - limit: 50
                      color: '#5BE12C'
                    - limit: 75
                      color: '#F5CD19'
                    - limit: 100
                      color: '#EA4228'
                  ticks:
                  min_value: 0
```

```
                  max_value: 100
                  unit: '%'
              - id: history
                type: history
                title: Последние 2 показания
                entity_id:
                  - humidity
                row-template: >
                  {%- set data = namespace(entities=[]) -%}
                  {%- for row in humidity -%}
                  {%- if loop.index <= 2 -%}
                  {%- set data.entities = data.entities +
[[row.time, row.state.value|string]] -%}
                  {%- endif -%}
                  {%- endfor -%}
                  {{ data.entities | tojson }}
                headers:
                  - Дата и время
                  - Влажность, %
```